A Simultaneous Bus Orientation and Bused Pin Flipping Algorithm

Fan Mo
Synplicity
fanmo@synplicity.com

Robert K. Brayton
University of California at Berkeley
brayton@eecs.berkeley.edu

Abstract — **The orientation of a bus is defined as the direction from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). Bused pin flipping is a property that allows several bused pins to flip without changing the system functionality. In this paper a simultaneous bus orientation and bused pin flipping algorithm is presented. The algorithm can be integrated into a bus-centric floorplanner targeting bus-rich designs such as microprocessors. Experimental results show that a floorplanner enhanced by the algorithm produces high quality floorplans in terms of bus routing.**

I. INTRODUCTION

Buses are used widely in computation and storage intensive designs like microprocessors [5]. Bus routing techniques have been studied by various researchers. Persky and Tran proposed a bus centric global routing algorithm that identified the topological commonality of different bits of the same bus and tried to re-use a common routing topology [11]. The idea of a common routing topology is adopted in our router and it is called the "virtual net". Without proper handling of bus orientations, even global routing of buses as in [11] is not always enough, since a detailed router may fail to implement the planned bus topologies because of twisting of bus bits and decreased accessibilities of the bused pins. If the detailed router must re-route some of the bus bits using different topologies, the advantages of using buses are lost. The simplified bus routing estimation techniques used in the floorplanning [1-4,6,7] make the routability and the bus routing quality of the final floorplan in doubt. Recently, a semi-detailed bus routing algorithm was proposed [9], which reported good quality and efficiency. This single bus routing algorithm focuses on the turning points in the virtual net and determines their orientations. Ignoring bus orientations can cause over-optimistic judgments of the floorplan results. Integrating a bus specific router in a floorplanner may boost the accuracy and quality of the results.

A useful characteristic of storage modules is bus pin flipping, that is, when a group of buses that connect to a storage module all flip their orientations, the functionality remains the same. For a storage module, the bused data input and output pins corresponding to the same internal storage core may form a "flipping group". Bused output pins for the read and write may form another flipping group. Flipping all the bused pins in the same flipping group, LSB becoming MSB, MSB becoming LSB etc, does not change the functionality and is totally transparent to other modules communicating with this module. This offers extra opportunity to successfully route certain buses, which otherwise are unroutable or cause twisting.

The simultaneous multiple bus orientation and bused pin flipping algorithm presented in this paper targets bus rich designs. Our algorithm, as opposed to the single bus router reported in [9], handles the routing of multiple buses and bused pin flipping at the same time. The algorithm can be used in the following scenarios: (1) as a standalone program that works on a given floorplan to handle all the buses; (2) as a bus routing procedure being frequently called during bus centric floorplanning; (3) as part of a full router to handle buses and bused pin flipping. In the next section, the problem of simultaneous multiple bus orientation and bused pin flipping is stated, along with definitions of terminologies. The algorithm is detailed in Section III. Experimental results are given in Section IV and Section V concludes.

II. TERMINOLOGIES AND PROBLEM FORMULATION

In this section, terminologies are defined and some basic concepts are presented, followed by a formulation of the simultaneous bus orientation and bused pin flipping problem.

Definition 1: A ***bused pin*** of an n-bit bus consists of n pins. The bits of the bused pin are arranged monotonically from the LSB to the MSB, and are equally spaced. We assume all bused bits use a uniform spacing. A bused pin is oriented horizontally or vertically. A horizontally oriented bused pin, or an ***X-pin***, can only be accessed from bottom and top. A vertically oriented bused pin, or a ***Y-pin***, can only be accessed from left and right.

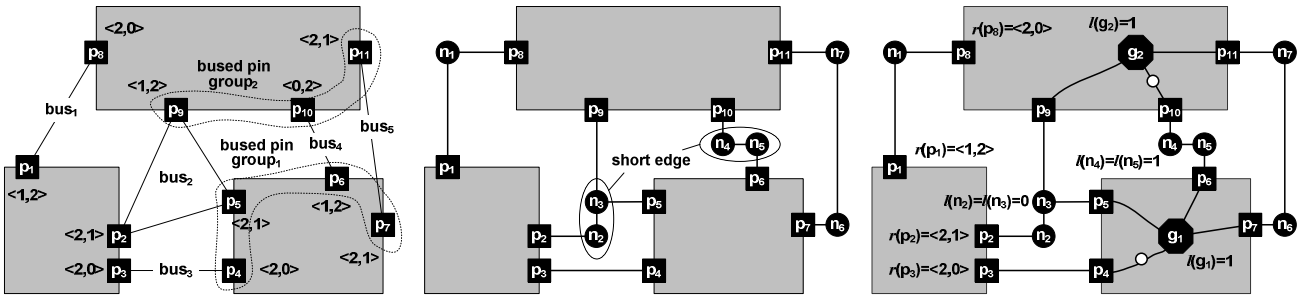
Definition 2: A ***virtual net*** is a single net that represents all the bits of a bus. A routed virtual net is composed of nodes and edges. ***Pin nodes*** correspond to bused pins. The rest of the nodes in the virtual net are called the ***routing nodes***. Another type of node, called a group node, will be introduced shortly in Definition 5.

Definition 3: The ***orientation*** of node d is defined as the direction from the LSB to the MSB, denoted by $r(d)=<r_X(d),r_Y(d)>$, where r_X(d) and $r_Y(d)$ are the orientations in X and Y directions, respectively. The binary-valued $r_X(d)$ and $r_Y(d)$ can be 0 for "negatively oriented" or 1 for "positively oriented". "Complement" operation ($\neg$) can be applied to these variables just as in normal Boolean operations. Value 2 is reserved for "undetermined" or "don't care".

Necessary condition for valid bus routing: If two nodes, d_1 and d_2, are connected by a horizontal edge, then $r_Y(d_1)=r_Y(d_2)$. If connected by a vertical edge, then $r_X(d_1)=r_X(d_2)$.

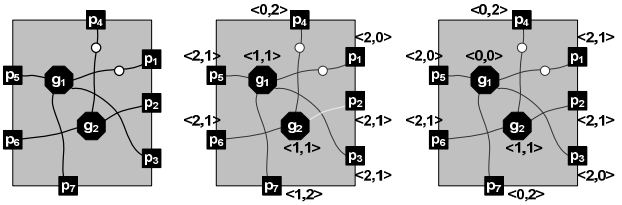
Definition 4: There are circumstances where either $r_X(d)=r_Y(d)$ or $r_X(d)=\neg r_Y(d)$ is enforced, which is called an ***interlocking*** and denoted by $l(d)=1$ or $l(d)=0$, respectively. If node d does not have interlocked X/Y orientations, $l(d)=2$. A type of interlocking, two turning nodes being connected by a short edge, was discussed in [9]. The concept of interlocking will be extended in the next definition.

Definition 5: A ***bused pin group***, or simply ***group***, is a set of bused pins of a module that can all ***flip*** together from their original orientations without changing the system functionality. A node with orientation $<r_X(d),r_Y(d)>$ will have $<\neg r_X(d), \neg r_Y(d)>$ after flipping. Figure 1(a) shows a module with two groups. A group has a ***group node***, represented by an octagon in the figure. All the bused pins of a group connect to the group node via edges. The edges are horizontal for a Y-pin, e.g. edge g_1-p_1, or vertical for an X-pin, e.g. edge g_1-p_7. Here "horizontal" and "vertical" are in some sense less strict though. Moreover, certain edges may carry a bubble, meaning that the orientation of the associated pin takes the complement orientation of



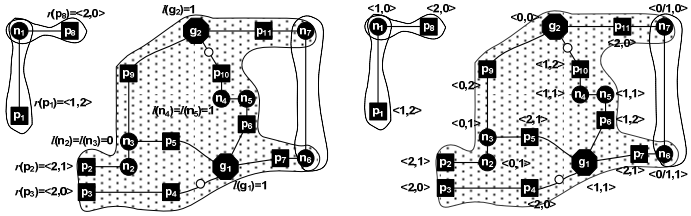
(a) the floorplan and bused pin groups (b) a forest of routed virtual net trees (c) the graph

Figure 2. An example



(a) two groups of a module (b) when $r(g_1)$=$r(g_2)$=<1,1> (c) when $r(g_1)$= <0,0> and $r(g_2)$=<1,1>

Figure 1. Bused pin groups and the group nodes of a module.



(a) edge sets (b) setting r_X's and r_Y's for the nodes

Figure 3. Edge set traversal and r_X/r_Y determination.

the group node. A group node can take an orientation of either <1,1> or <0,0>, meaning all associated pin nodes take their original or flipped orientations, respectively. This is another type of interlocking, where $l(d)$=1. Figure 1(b) and (c) show how the orientations of the group-nodes determine the orientations of the pin nodes in their groups.

Definition 6: After globally routing all the buses using their virtual nets, we obtain a forest of routing trees. Next, each bused pin group contributes a group node, and the node is connected to the pin nodes in the group, thus connecting up some trees. Now we have a ***graph*** with ***node set D*** and ***edge set E***. $D = D_{PG} \cup D_{PNG} \cup D_N \cup D_G$, where where D_{PG} is the set of pin nodes belonging to a bused pin group, D_{PNG} are the pin nodes that belong to no group, D_N is the set of the routing nodes, D_G is the set of the group nodes. Obviously they are disjoint. For $d \in D_{PG} \cup D_{PNG}$, $r_Y(d)$=2 for X-pins, and $r_X(d)$=2 Y-pins. For $d \in D_{PNG}$, the value of $r_X(d)$ or $r_Y(d)$ will be determined once a floorplan is given; while for $d \in D_{PG}$, the value of $r_X(d)$ or $r_Y(d)$ needs to be determined by the algorithm.

An example is given in Figure 2. Figure 2(a) is a floorplan of three modules, five buses and two bused pin groups. The original orientations of the bused pins are also shown. The routed virtual nets of the buses are shown in Figure 2(b), where five separate trees form a forest. When taking into account the bused pin groups, as shown in Figure 2(c), group nodes are added and some trees are now connected and cycles appear. The two group nodes are both given interlocked property $l(d)$=1. The two short routing edges in Figure 2(b) are also given interlocking properties, based on Definition 4. Non-grouped pin nodes, p_1, p_2, p_3 and p_8, have their orientations fixed. The original orientations of the grouped pin nodes, p_4, p_5, p_6, p_7, p_9, p_{10} and p_{11}, determine whether or not a bubble should be added to the edges connecting them to the group node. The algorithm works with the graph in Figure 2(c).

Problem statement: Find orientations for each node, such that the necessary condition is satisfied everywhere.

Theorem 1: The simultaneous bus orientation and bused pin flipping problem is feasible if and only if

- For any horizontal edge $<d_i,d_j>$, $r_Y(d_i)=r_Y(d_j)$.
- For any vertical edge $<d_i,d_j>$, $r_X(d_i)=r_X(d_j)$.
- If $l(d)$=0 for node d, $r_X(d)=\neg r_Y(d)$.
- If $l(d)$=1 for node d, $r_X(d)=r_Y(d)$.

The theorem is straightforward by construction and the proof is omitted.

III. ALGORITHM

We seek a partition of the graph, such that within each partition, all nodes have their $r_X(d)$'s and/or $r_Y(d)$'s dependent of each other. The graph can be partitioned into disjoint edge sets, $E=E_1 \cup E_2 \cup \ldots E_n$ using these relations. The partition is unique. Note that a node may appear in two edge sets. For node d with at least one edge in E_i, there must be d_2 that also has one edge in E_i, such that $r_X(d)$ or $r_Y(d)$ relates to $r_X(d_2)$ or $r_Y(d_2)$. Using the example in Figure 2, an edge set is depicted as a dotted region in Figure 3(a). In total, there are four edge sets in this example. In the following discussion, variable v is always used to represent a direction of either X or Y. $\neg v$ means taking the orthogonal direction of v. We introduce the concept of "referencing". $r_{vi}(d_i) \propto r_{vj}(d_j)$ means $r_{vi}(d_i)$ references $r_{vj}(d_j)$, although the value of $r_{vj}(d_j)$ may not have been decided; once the value of $r_{vj}(d_j)$ is known, then the reference becomes assignment, $r_{vi}(d_i)=r_{vj}(d_j)$. Complement can occur in the expression. For instance, $r_{vi}(d_i) \propto \neg r_{vj}(d_j)$ means that if one takes the value of 0 the other must take 1. References are also used to indicate if node d_i has been traversed for X and Y orientations. If not, $r_X(d_i) \propto$null and $r_Y(d_i) \propto$null, respectively. On the other hand, self referencing, $r_{vi}(d_i) \propto r_{vi}(d_i)$, means a new round of traversing starts from d_i. Obviously referencing is transitive and symmetric (with complement).

With referencing, we can start from any node d_0 and travel along edges to other nodes, making their r_X or r_Y reference those of previously traversed ones. Since referencing is transitive, this implies that all the traversed nodes reference (possibly with complement) $r_{v0}(d_0)$, where v_0 is X or Y. The traversal may stop at nodes, which have independent r_X and r_Y, e.g. routing nodes n_6 and n_7 in Figure 2 and 3. The traversal may involve re-convergence, where a previously traversed node is encountered. In this case, the reference already set on the node must be the same as the reference to be set. If a conflict occurs, the problem is unsatisfiable. If a non-grouped pin node is encountered, its known orientation immediately sets the value of

$r_{v0}(d_0)$ through referencing. When more than one non-grouped pin node is encountered, they both set the value of $r_{v0}(d_0)$. If these settings conflict, the problem is unsatisfiable. A round of traversal ends, when no additional edge can be included such that the referencing relation can be further propagated. The traversed edges form one edge set. If no non-grouped pin node has been reached in the edge set, $r_{v0}(d_0)$ can be either 0 or 1, leading to two configurations. Otherwise there is only one configuration for the set. The entire problem contains 2^{N_U} configurations, where N_U is the number of edge sets containing no non-grouped pin nodes. Hence, the algorithm not only tells whether the problem is feasible or not, but also tells how many different configurations exist and what they are.

Initially all nodes have their references set to null, indicating that none have been traversed. A round of traversing starts from any node d_0 that has $r_X(d_0)\propto$null or $r_Y(d_0)\propto$null. Suppose $r_Y(d_0)\propto$null, hence edge direction X is chosen, and $r_Y(d_0)$ becomes something that all following nodes reference, possibly with complement. Starting from d_0, we keep visiting nodes connected by horizontal edges. When reaching node d from d_{PREV}, $r_Y(d)$ copies the reference of d_{PREV} with complement if the edge carries a bubble. If $r_Y(d)$ has already had a reference, then a re-convergence occurs and the two references must be the same. Otherwise a conflict is detected. If $d\in D_{PNG}$, whose value of $r_Y(d)$ is already known, then $r_Y(d_0)$ needs to take the same value with possible complement. If $r_Y(d_0)$ is already assigned a value, d_0 itself or some previously reached node being a non-grouped pin node, we check if there is a conflict. If an interlocking node is reached, $l(d)\neq 2$, reference is relayed from $r_Y(d)$ to $r_X(d)$. We insert d and edge search direction Y in a queue. When all nodes along horizontal edges have been traversed, the queue may contain some nodes that require the traversal to continue in the orthogonal direction, in this case the Y direction. We keep traversing these, possibly with further switching in traversing direction, until the queue is empty. This terminates the traversal of one edge set. Picking another $r_X(d_0)\propto$null or $r_Y(d_0)\propto$null, we then work with another edge set. The algorithm completes when all orientations have references. When feasibility is achieved, we also know all feasible configurations. For edge sets where $r_X(d_0)$ or $r_Y(d_0)$ have not been set, we can arbitrarily set them to 0 or 1. Then all the non-starting nodes can derive their orientation values through the references that are established during the traversal. Figure 3(b) shows a feasible configuration achieved by the algorithm. Note that edge set composed solely of the edge between n_7 and n_8 has two configurations, that is, $r_X(n_7)=r_X(n_8)=0$ or 1. In fact all other edge sets in the example have a single configuration. Once all nodes have their orientations determined, the realization of the buses is trivial.

The complexity of the algorithm is $O(|D|+|E|)$. Since a routing node can have at most 4 incident edges and a group node's degree is also limited by some constant number, the complexity can be written as $O(|D|)$. This is much more efficient than enumerating all combinations of bused pin flipping and then running an algorithm like [9] for each bus. Under certain circumstances, we may not want the algorithm to fail immediately after a conflict is detected. When the algorithm is integrated into a simulated annealing based floorplanner, for instance, it is desirable to have a more gradual penalty for failures in the cost function. Therefore, when a conflict occurs at a node, we simply mark the bus associated with the node as failed and continue (if it is a group node, more than one bus is marked as failed). Finally the number of failed buses is returned.

IV. EXPERIMENTAL RESULTS

We established an experimental flow so that different bus algorithms could be plugged in for comparison. The flow contained two parts: bus centric floorplanning and full detailed routing. The sequence pair method was used for floorplan representation [8]. Various bus routing algorithms [11,3,9] including ours can be plugged in to the floorplanner to handle buses. The cost function used in the simulated annealing was

$$cost=a_U\times\mathrm{Max}(0,L_{UTIL}-Utilization)+a_{ASP}\times\mathrm{Max}(0,AspectRatio-L_{ASP})+a_W\times TotalWireLength+a_F\times TotalBusFailure+a_C\times TotalBusCongestion,$$

in which, a_U=0.3, a_{ASP}=0.1, a_W=0.45, a_F=0.1 and a_C=0.05 are the weights associated with the corresponding terms. L_{UTIL}=0.8 is the target utilization and L_{ASP}=1.1 is the limit for the aspect ratio of the floorplan. The total wire length includes the estimated wire lengths of both the non-bused nets and the buses.

Four bus routers were compared using this same floorplanning framework: the virtual-net (abbr. *v-net*) algorithm [11], the 0/1/2-bend (abbr. *bend*) algorithm [3], the single bus orientation (abbr. *bus-or*) algorithm and our algorithm (abbr. *this*). The single bus orientation approach includes bused pin flipping as a random move in the annealing. The 0/1/2-bend approach ignores all routing blockages. All failed buses were demoted to individual nets. A simple bus congestion removal and estimation procedure was also implemented. In addition, a post processing for the final floorplan was provided, which is basically a run of our algorithm including virtual net routing and congestion removal. This gives a standardized way to measure bus failures for comparison purposes. A commercial router was used to complete the routing. The single-bus orientation algorithm [9] and our algorithm are both able to forward annotate successfully routed buses. The experimental flow was tested on an AMD 1.6GHz machine. Five test cases, extracted from microprocessor designs, were used in the experiment. The characteristics of these test cases are given in Table I.

The area, run time and bus failure results are also given in Table I. All four algorithms produced very similar areas, which was a natural result because we only penalized utilizations below L_{UTIL}. In terms of the run time, the virtual-net and 0/1/2-bend algorithms were marginally faster than our algorithm, while all three were about 15% faster than the single-bus orientation algorithm. Our algorithm achieved zero failures for all the test cases. The virtual-net algorithm always reported zero bus failures, because it only routed the virtual nets and this never failed. The 0/1/2-bend algorithm reported bus failures for the last two test cases using its own standard (a bus failed when it could not be constructed using 0/1/2-bend topologies), however our algorithm judged that four designs had bus failures.

The first part of Table II shows the total wire lengths, including non-bused nets and buses, estimated by the algorithms. Directly comparing the numbers is somewhat misleading. The real routing results, obtained from the commercial router, show that the floorplan generated using our algorithm had 6%, 25% and 3% shorter real wire length than the other three, respectively. Comparing the wire lengths estimated during floorplanning and after real routing, the 0/1/2-bend based floorplanner severely underestimated this, while the single-bus orientation algorithm and our algorithm gave fairly good estimations. In terms of via count, our algorithm produced 93%, 118% and 1% fewer than the other three algorithms, respectively. To measure bus regularity, we used the bitwise wire length mismatch, that is, the maximum percentage difference between the wire length of any bit of a bus and the average wire length of all the bits. The mismatch of our algorithm is 8%, 6% and 0.3% smaller than those of the other three algorithms, respectively. It is worth mentioning that among the three algorithms in comparison, the single-bus orientation algorithm [9] is the "closest" to ours. However, results have shown that lacking the ability to directly handle bused pin flipping during bus routing makes it 3%, 1% and 0.3% worse than our algorithm in terms of final wire lengths, via count, and bitwise mismatch. There were tests in the experiment, for which it ended up with failed buses while our algorithm succeeded. Also, it took 14% longer run time than ours.

Table I. Floorplan results

design	#module	#routing blockages	#non-bus nets	# buses	# bused nets	# pin groups	area (mm^2) v-net	area (mm^2) bend	area (mm^2) bus-or	area (mm^2) this	total bus failures v-net *	total bus failures bend	total bus failures bus-or	total bus failures this	run time (sec) v-net	run time (sec) bend	run time (sec) bus-or	run time (sec) this
C1	19	1	117	36	551	26	0.86	0.86	0.87	0.86	(6)**	0 (3)**	0 (0)**	0	140	143	174	141
C2	19	1	174	34	790	17	1.03	1.02	1.02	1.02	(3)	0 (0)	0 (0)	0	222	218	225	220
C3	19	1	166	34	994	17	1.18	1.17	1.13	1.17	(8)	0 (1)	0 (0)	0	231	180	230	218
C4	25	4	160	35	842	30	260	260	259	261	(23)	1 (22)	3 (1)	0	650	766	899	707
C5	58	3	475	91	2357	46	23.4	23.0	22.5	23.1	(61)	4 (50)	3 (1)	0	2571	2622	2946	2603
average							101%	100%	99%	**100%**					99%	98%	114%	**100%**

* The v-net algorithm does not provide #bus failures. ** Estimated by running a post-processing using our algorithm.

Table II. Detailed routing results for floorplan *without* post-processing (bus-or and this forward annotated successful buses)

design	total estimated wire length (mm) v-net	bend *	bus-or	this	total real wire length (mm) v-net	bend	bus-or	this	total real via number v-net	bend	bus-or	this	ave. bus-bit wire length mismatch v-net	bend	bus-or	this
C1	275	229	293	290	283	286	304	293	3209	3318	1811	1799	8.2%	7.5%	2.5%	2.7%
C2	380	378	359	353	396	558	374	370	4750	6315	2595	2556	9.9%	11%	5.4%	5.6%
C3	488	591	529	498	571	799	560	526	6341	7810	3235	3200	13%	11%	5.2%	4.4%
C4	4986	5420	4496	4517	5204	5551	4549	4531	6095	6273	3354	3412	9.9%	7.0%	1.6%	1.1%
C5	4429	3687	4392	4408	4532	4507	4468	4439	20893	21385	9454	9337	12%	8.6%	2.4%	1.9%
average	102%	102%	102%	**100%**	106%	125%	103%	**100%**	193%	218%	101%	**100%**	11%	9.0%	3.4%	3.1%

* The 0/1/2-bend algorithm did not obey routing blockages and bus pin locations. When estimating bus routing lengths, the 0/1/2-bend topologies were both broken into 0-bend stripes. We also assumed that the 0-bend stripes used the shortest possible lengths to fulfill the connections.

Table III. Detailed routing results for floorplan *with* post-processing (all algorithms, after post-processing) forward annotated successful buses)

design	total estimated wire length (mm) v-net	bend	bus-or	this	total real wire length (mm) v-net	bend	bus-or	this	total real via number v-net	bend	bus-or	this	ave. bus-bit wire length mismatch v-net	bend	bus-or	this
C1	291	285	293	290	292	289	304	293	2786	3004	1811	1799	6.6%	5.9%	2.5%	2.7%
C2	397	489	359	353	405	526	374	370	3854	4734	2595	2556	7.3%	6.2%	5.4%	5.6%
C3	529	722	529	498	539	741	560	526	5318	4395	3235	3200	10%	6.9%	5.2%	4.4%
C4	5133	5489	4496	4517	5158	5503	4539	4531	5496	5968	3394	3412	8.5%	6.6%	1.4%	1.1%
C5	4508	4301	4392	4408	4546	4448	4455	4439	17470	16593	9436	9337	9.0%	7.1%	2.3%	1.9%
average	107%	120%	102%	**100%**	106%	121%	102%	**100%**	164%	168%	101%	**100%**	8.3%	6.5%	3.4%	3.1%

Having shown that integrating a more precise bus router into the floorplanner produced better final bus routes, we were interested in knowing how a post-processing step alone using our algorithm could help. We took the floorplans generated by the first three algorithms and ran our algorithm once (this had been used to report bus failures in Table I). All runs of the post-processing took less than one second. The floorplans were passed to the commercial router with successfully routed buses forward annotated. Table III gives the results for such an experiment. The results of the real wire lengths of the three still lagged behind ours, although small improvements did exist. The via counts as well as the bus-bit wire length mismatch were also remarkably reduced for the virtual-net and 0/1/2-bend algorithms. This indicates that integrating the bus router, assuming it is fast enough, into the floorplanner perhaps is the best solution. Post processing alone helps, but may not help enough. On the other hand, for the first two algorithms, the estimated wire lengths became much closer to the real wire lengths. So our algorithm can be very useful for evaluating the quality of existing floorplans in terms of bus routing. For example, a layout editor for manual floorplanning can integrate our algorithm to give real-time evaluation of the floorplan in terms of bus routing quality.

V. CONCLUSION

An algorithm for simultaneously determining bus orientation and bused pin flipping was presented. The algorithm deals with real bus pin positions and routing blockages. The flexibility of bused pin flipping was fully exploited. The efficient linear time algorithm can be integrated into a bus centric floorplanner tailored for bus intensive designs. A floorplanner enhanced by the algorithm produces high-quality bus planning while achieving normal floorplanning goals like area and aspect ratio. Experimental results show that the algorithm outperforms other existing bus routing algorithms in terms final wire length, via count and routing regularity.

REFERENCES

[1] F.Rafiq, M.Chrzanowska-Jeske, H.H.Yang and N.Sherwani, "Integrated Floorplanning with Buffer/Channel Insertion for Bus-Based Microprocessor Designs", *ISPD* 2002, pages 56-61.

[2] H.Xiang, X.Tang and D.F.Wong, "Bus-Driven Floorplanning", *ICCAD* 2003, pages 66-73.

[3] J.H.Y.Law and E.F.Y.Young, "Multi-Bend Bus Driven Floorplanning", *ISPD* 2005, pages 113-120.

[4] T.C.Chen and Y.W.Chang, "Modern Floorplanning based on Fast Simulated-Annealing", *ISPD*, 2005, pages 104-112.

[5] S. Iwata, et al., "Performance Evaluation of a Microprocessor with On-Chip DRAM and High Bandwidth Internal Bus", *IEEE CICC*, May 1996, pages 269-272.

[6] E.F.Y.Young, C.C.N.Chu and M.L.Ho, "A Unified Method to Handle Different Kinds of Placement Constraints in Floorplan Design", *ASPDAC* 2002, pages 661.

[7] X.Tang and D.F.Wong, "Floorplanning with Alignment and Performance Constraint", *DAC* 2002, pages 848-853.

[8] H.Murata, K.Fujiyoshi, S.Nakatake and Y.Kajitani, "Rectangle-Packing based Module Placement", *ICCAD* 1995, pages 472-479.

[9] F. Mo and R.K. Brayton, "A Semi-Detailed Bus Routing Algorithm with Variation Reduction", *ISPD* 2007, pages 143-150.

[10] X. Tang and D.F.Wong, "FAST-SP: A Fast Algorithm for Block Placement based on Sequence Pair", *ASPDAC* 2001, pages 521-526.

[11] G. Persky and L.V. Tran, "Topological Routing of Multi-Bit Data Buses", *DAC* 1984, pages 679-682.